「小型マイコンモジュール」
ＡＴ８０１２
取　扱　説　明　書

株式会社　エーシーティー・エルエスアイ

Ｒ-1.31

本資料掲載の技術情報及び半導体のご使用につきましては以下の点にご注意願います。

改訂履歴

Rev.	Date	改　訂　内　容	備　　考
1.0	01/08/26	初　版	
1.1	02/06/20	サンプルソース追加　他	
1.2	02/07/10	端子表全面改訂	
1.3	02/07/13	モード端子状態改定	
1.31	03/09/01	メモリーマップ変更	

は じ め に

ＡＴ８０１２はアナログデータロガーシリーズとして開発した弊社製品です。Ｈ８マイコンの機能を有効に利用するために、出来る限り、Ｈ８の端子を外部に出力し汎用のマイコンモジュールとしても機能するように設計されています。
Ｈ８マイコンに関する仕様は記述しておりません。また、ＲＴＣ、フラッシュメモリーなど周辺ＩＣの詳細については、この説明書では解説しておりません。ご利用になる場合は各メーカーの仕様書を入手の上ご利用ください。

H8 マイコンの概要

ＡＴ８０１２に搭載された、株式会社日立製作所製のマイコンＨ８／３００Ｈシリーズ（以下Ｈ８マイコン）は１６ビット×１６個の汎用レジスタをもち、１６Ｍバイトのリニアなアドレス空間を利用できます。また、周辺機能が豊富で１６ビットタイマー、プログラマブルタイミングパターンコントローラ、ウォッチドッグタイマー、シリアルコミュニケーションインタフェース、１０ビットＡ／Ｄ変換器、８ビットＤ／Ａ変換器、ＤＭＡコントローラ、リフレッシュコントローラなどを内臓しています。

機能の詳細については
株式会社日立製作所発行のマニュアルを参照してください。

機能の詳細については
株式会社日立製作所発行のマニュアルを参照してください。

・ Ｈ８／３０４２シリーズ ハードウェアマニュアル
・ Ｈ８／３００Ｈシリーズ プログラミングマニュアル ｅｔｃ．

目　　　　　次

ページ

第１章　概　要

ＡＴ８０１２は１６ビットマイコン「Ｈ８／３０４２」を搭載したマイコンモジュールです。標準仕様で１ＭビットＳＲＡM及び、４ＭビットフラッシュＲＯＭ（内ユーザー領域は３８４Ｋバイト）を搭載し、オプションで株式会社リコー製リアルタイム・クロック「RS5C317A」及び、ＮＡＮＤ型フラッシュメモリー（２Ｍ、４Ｍ、８Ｍバイト）が搭載可能となっています。

Ｈ８マイコンの豊富な機能をそのまま利用できるように５０ピンの峡ピッチコネクタを２個使用し、すべて機能ポートを外部に直接接続できるようにしてあります。

ユーザーが開発したプログラムはRS-232C通信ターミナルソフト（XMODEMプロトコル）で転送、書き換えが可能になっています。

１．１ 構成

ＡＴ８０１２には標準で１MビットのＳＲAMと４MビットのフラッシュＲOMが搭載されています。

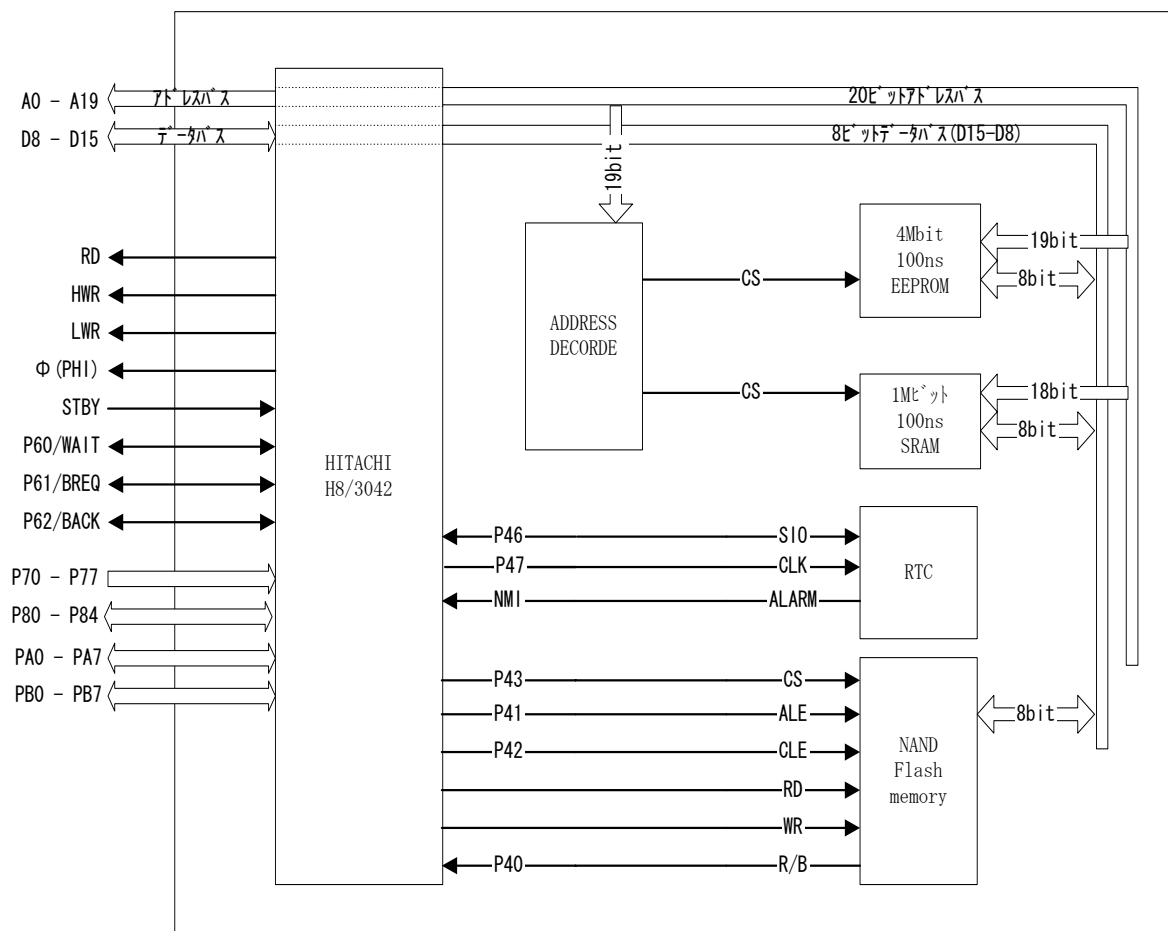
フラッシュＲOMの一部はシステムで使用していますので、ユーザーが利用できるプログラム領域は３８４Kバイトになっています。

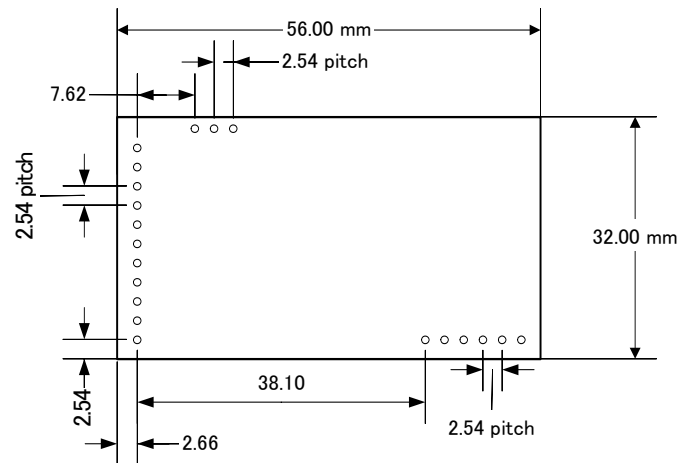
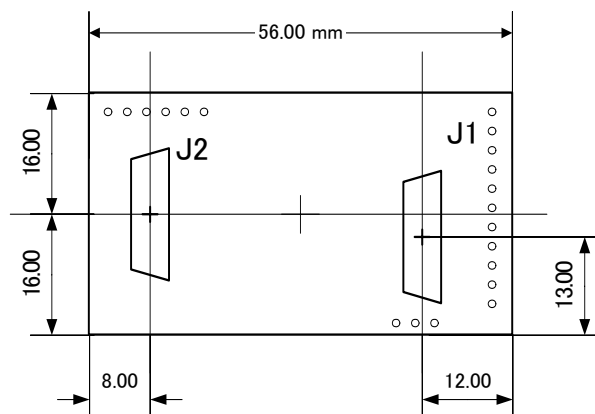
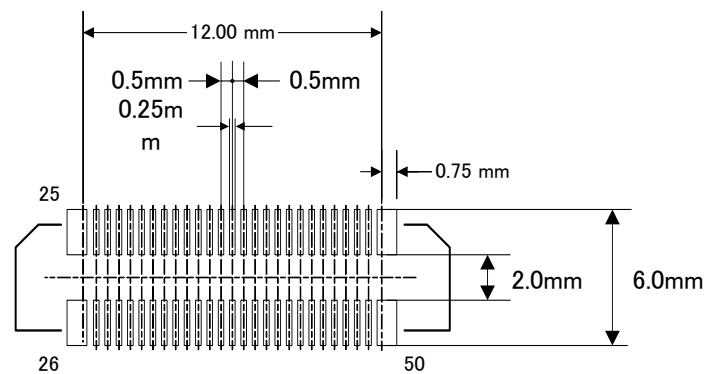
リアルタイム・クロックはアラーム機能付きで、アラーム出力がＨ８のNMＩ端子に、定周期割込み出力がＩＲＱ０端子に接続されていますので低消費電力状態からの復帰に利用できます。

NAND型フラッシュメモリーは株式会社東芝製 「ＴＣ５８１６ＦＴ」または「ＴＣ５８３２ＦＴ」相当品を搭載します。このメモリーはそれぞれ２M、４M、８Mバイトのメモリー容量を持ち、主にストレージメモリーとして利用します（プログラムメモリーとしては利用できません)。

ＡＴ８０１２の構成図を図１に示します。

図 1　AT8012 構成図

ＡＴ８０１２ ブロック図

１.２ 概 観

基板外形及び嵌合コネクタのプリント基板パターン図を以下に示します。

裏面図

嵌合コネクタ基板パターン

１．３ 端子構成

　ＡＴ８０１２はＨ８の端子を可能な限り外部へ出すことを考慮し、外部接続コネクタに松下電工株式会社製 峡ピッチ多極コネクタ 「ＡＸＫ５Ｓ５００４５」を採用しました。これに嵌合するコネクタは同社製 「ＡＸＫ６Ｓ５０５４５Ｐ」となります。概観図にプリント基板パターン図を添付しました。端子番号もその図を参照してください。
　概観図にプリント基板パターン図を添付しました。端子番号もその図を参照してください。 以下にAT8012 の峡ピッチコネクタの端子表を示します。信号名に＊がついているものは H8 マイコンに直接接続されています。

CN1 端子表　　外形図（Ｊ2）

端 子 番 号	端 子 名	I/O	説 明
25,26	VIN	I	電源入力（5．5V～15V）
2,49	VBU	I	RTCバックアップ電源（1．5V～5V）
1,50	VDD	O	電源出力（＋5V）
7,33,35,44	GND	I	グランド
22-24,27-29, 31,32,34	NC		何も接続しないでください。
16	$\overline{\text{IOE0}}$	O	I/OまたはｱﾄﾞﾚｽﾃﾞｺｰﾄﾞI信号(B0000 H – BFFFF H)
17	$\overline{\text{IOE1}}$	O	I/OまたはｱﾄﾞﾚｽﾃﾞｺｰﾄﾞI信号(C0000 H – CFFFF H)
18	$\overline{\text{IOE2}}$	O	I/OまたはｱﾄﾞﾚｽﾃﾞｺｰﾄﾞI信号(D0000 H – DFFFF H)
21	$\overline{\text{IOE3}}$	O	I/OまたはｱﾄﾞﾚｽﾃﾞｺｰﾄﾞI信号(E0000 H – EFFFF H)
36	*A15	O	ｱﾄﾞﾚｽﾊﾞｽ
15	*A14	O	ｱﾄﾞﾚｽﾊﾞｽ
37	*A13	O	ｱﾄﾞﾚｽﾊﾞｽ
14	*A12	O	ｱﾄﾞﾚｽﾊﾞｽ
38	*A11	O	ｱﾄﾞﾚｽﾊﾞｽ
13	*A10	O	ｱﾄﾞﾚｽﾊﾞｽ
39	*A9	O	ｱﾄﾞﾚｽﾊﾞｽ
12	*A8	O	ｱﾄﾞﾚｽﾊﾞｽ
40	*A7	O	ｱﾄﾞﾚｽﾊﾞｽ
11	*A6	O	ｱﾄﾞﾚｽﾊﾞｽ
41	*A5	O	ｱﾄﾞﾚｽﾊﾞｽ
10	*A4	O	ｱﾄﾞﾚｽﾊﾞｽ
42	*A3	O	ｱﾄﾞﾚｽﾊﾞｽ
9	*A2	O	ｱﾄﾞﾚｽﾊﾞｽ

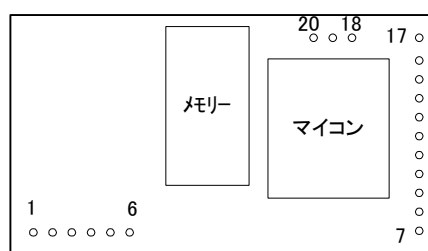
端 子 番 号	端　子　名	I/O	説　　　明
43	*A1	O	ｱﾄﾞﾚｽﾊﾞｽ
8	*A0	O	ｱﾄﾞﾚｽﾊﾞｽ
20	*HWR	O	D8～D15書込ｽﾄﾛｰﾌﾞ信号
19	*RD		読み込みｽﾄﾛｰﾌﾞ信号
30	RES	I	ｼｽﾃﾑﾘｾｯﾄ
45	*D15	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD7）
6	*D14	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD6）
46	*D13	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD5）
5	*D12	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD4）
47	*D11	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD3）
4	*D10	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD2）
48	*D9	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD1）
3	*D8	I/O	ﾃﾞｰﾀﾊﾞｽ（AT8012内部ではD0）

CN2 端子表　　外形図（Ｊ1）

端子番号	端　子　名	I/O	説　　　明
26,27	VDD	O	電源出力（＋5V）
36,42	GND	I	グランド
24,25	NC		何も接続しないでください
1	PRES	O	リセット出力
2	*PB7	I/O	H8 Port B7端子
3	*PB6	I/O	H8 Port B6端子
4	*PB5	I/O	H8 Port B5端子
5	*PB4	I/O	H8 Port B4端子
6	*PB3	I/O	H8 Port B3端子
7	*PB2	I/O	H8 Port B2端子
8	*PB1	I/O	H8 Port B1端子
9	*PB0	I/O	H8 Port B0端子
10	*TXD0	I/O	H8 TXD0端子
11	*TXD1	I/O	H8 TXD1端子
12	*RXD0	I/O	H8 RXD0端子
13	*RXD1	I/O	H8 TXD1端子
14	MDP (*P94)	I	システムモード設定端子(この端子はMDP用途以外に使用しないでください)
15	COM (*P95)	I	モード設定端子

端子番号	端　子　名	I/O	説　　　明
16	*NMI	I	H8 NMI端子
17	*WAIT	I/O	H8 WAIT端子
18	*BREQ		H8 BREQ端子
19	*BACK		H8 BACK端子
20	*PHI		H8 PHI端子
21	*STBY		H8 STBY端子
22	32KOUT	O	RTC調整用32KHz出力端子
23	CLKE	I	32KOUT出力イネーブル端子
28	*P70		H8 Port 70端子
29	*P71		H8 Port 71端子
30	*P72		H8 Port 72端子
31	*P73		H8 Port 73端子
32	*P74		H8 Port 74端子
33	*P75		H8 Port 75端子
34	*P76		H8 Port 76端子
35	*P77		H8 Port 77端子
37	*P80/IRQ0		H8 Port 80端子
38	*P81/IRQ1		H8 Port 81端子
39	*P82/IRQ2		H8 Port 82端子
40	*P83/IRQ3		H8 Port 83端子
41	*P84		H8 Port 84端子
43	*PA0		H8 Part A0端子
44	*PA1		H8 Part A1端子
45	*PA2		H8 Part A2端子
46	*PA3		H8 Part A3端子
47	*PA4		H8 Part A4端子
48	*PA5		H8 Part A5端子
49	*PA6		H8 Part A6端子
50	*PA7		H8 Part A7端子

峡ピッチコネクタ以外の端子は主に電源、アナログ入力などで、アナログデータロガーとして用いる場合、ＡＴ８０１２単体で構成できる様になっています。
下図に端子構成を示します。

表面図

図中の数字は端子番号を示します。
各端子の詳細を下表に示します。

No	名　　称	説　　　　明
1	VS	デジタルグランド
2	VB	ＲＴＣバックアップ電源（1Ｖ～5Ｖ）
3	VO	デジタル電源出力（＋５Ｖ）
4	RS	リセット入力
5	VS	グランド
6	VI	電源入力（6Ｖ～15Ｖ）
7	VO	アナログ用電源出力（+5Ｖ）
8	A0	アナログ入力０
9	A1	アナログ入力１
10	A2	アナログ入力２
11	A3	アナログ入力３
12	VS	アナログ用グランド
13	I0	割込０
14	I1	割込１
15	MD	システム／ユーザーモード切り替え
16	CM	通信／ロギングモード切り替え
17	VC	デジタル電源出力（＋５Ｖ）
18	TX	シリアル送信
19	RX	シリアル受信
20	VS	デジタルグランド

１．４ アドレスマップ

ＡＴ８０１２のモード別アドレスマップを以下に示します。

図 2　アドレスマップ１

第２章　動作モード

　ＡＴ８０１２はユーザーのプログラムが動作するユーザーモードとユーザーのプログラムの書き換えなどを行うシステムモードに大別されます。
　モードの切り替えはＣＯＭ端子とＭＤＰ端子の２本の端子で行い、３つのモードに切り替わります。

　H8 マイコンにも動作モードがありますが、ＡＴ８０１２では H8 のモード５で動作します。

　モード端子とモードの設定を表１に示します。

表 1 モード設定表

モード端子状態 (1 で High ﾚﾍﾞﾙ)		モード名	説　　明
MDP	COM		
0	0	ﾕｰｻﾞｰﾓｰﾄﾞ 0	H8-CPU のﾚｼﾞｽﾀを初期化後、無条件にｱﾄﾞﾚｽ 10400H番地にｼﾞｬﾝﾌﾟします。
0	1	ﾕｰｻﾞｰﾓｰﾄﾞ 1	ﾃﾞｰﾀﾛｶﾞｰ機能用のﾀｰﾐﾅﾙﾓｰﾄﾞです。
1	1	ｼｽﾃﾑﾓｰﾄﾞ	ﾌﾟﾛｸﾞﾗﾑ書き換えなどを行います。

モード端子は開放状態で Low ﾚﾍﾞﾙ　になっています。

第３章　コマンド説明

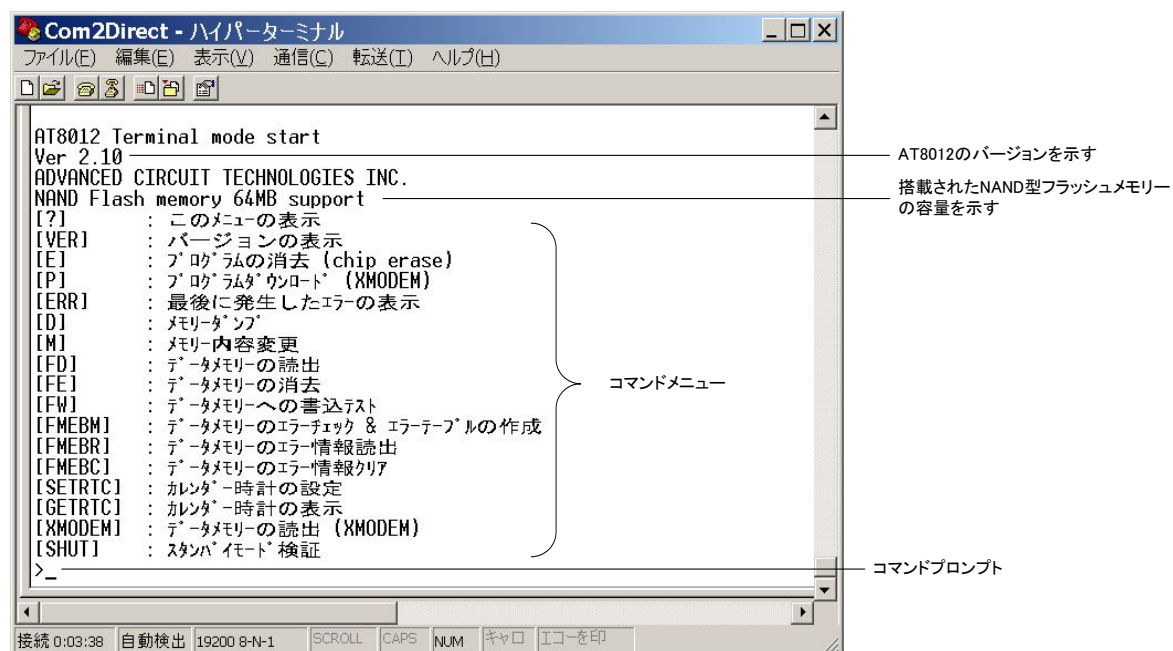
　ＡＴ８０１２はＨ８マイコンのＳＣＩのチャンネル０を利用し、プログラムの変更ができるようになっています（XＭｏｄｅｍ（チェックサム）プロトコルが利用できるターミナルソフトをご利用ください)。ＡＴ８０１２のＴＸＤ０、ＲＸＤ０端子をＲＳ２３２Ｃレベルコンバーターを介してパソコンのＲＳ２３２Ｃコネクタに接続します。パソコンのターミナルソフトを起動し、19200bps、ストップビット1、パリティーなし、データ長８ビットに設定します。

通信条件

データ伝送方式	調歩同期式
通信方式	全二重
電気的条件	RS-232C（AT8012 は TTL レベル３線式）
通信プロトコル	無手順（キャラクタ電装）
通信速度	19200bps
ストップビット	１ビット
データ長	8 ビット
パリティー	なし

モードをシステムモードになるようにモード端子を設定し、電源をＯＮにしてください。

上記の手続きを行うと次画面の様にメニューが表示されます。

画面Ａ

3. 1 コマンドの一覧

プログラムの書き換えを説明する前にメニューに表示されているコマンド一覧を記載します。

	コマンド	機　　能
1	?	コマンドの一覧を表示します
2	VER	本装置のバージョンを表示します
3	E	ユーザープログラムを消去します
4	P	ユーザープログラムを書き込みます
5	ERR	最後に発生したエラーを表示します（プログラム書込み時のエラー）
6	D	メモリー内容のダンプを表示します
7	M	メモリー内容の編集を行います
8	FD	NAND 型フラッシュメモリーの内容をダンプします
9	FE	NAND 型フラッシュメモリーの内容を消去します
10	FW	NAND 型フラッシュメモリーにデータを書き込みます （１ページ分のデータフィル）
11	FMEBM	NAND 型フラッシュメモリーの不良ブロックを検出し、テーブル化します
12	FMEBR	不良ブロックテーブルの読み出しを行います
13	FMEBC	不良ブロックテーブルを消去します
14	SETRTC	内蔵 RTC の設定を行います
15	GETRTC	内蔵 RTC の読み出しを行います
16	XMODEM	NAND 型フラッシュメモリーの内容を XMODEM プロトコルで読み出します （不具合あり。使用しないでください）
17	SHUT	低消費電力モードのテスト用

３．２ コマンド説明

ＡＴ８０１２はのターミナル画面のコマンドの入力はＥｎｔｅｒキーで受け付けるようになっています。コマンドとともにＥｎｔｅｒキーを入力してください。

コマンド説明中に<CR>と表記されている部分は Enter キーの入力を意味します。

【？】: Command help

■ **説明**

コマンドメニューの表示

■ **フォーマット**

?<CR>

■ **例**

【E】: ﾌﾟﾛｸﾞﾗﾑの消去（chip erase)

■ 説明

プログラムを書き込むメモリの消去を行います。
プログラムを書き換える時、一度メモリを消去してください。

■ フォーマット

Ｅ

■ 例

```
>E<CR>
Program erase ? (Y/N)
Erase done
>
```

“Ｅ”キーを押下すると“Ｐｒｏｇｒａｍ　ｅｒａｓｅ？（Ｙ／Ｎ)”メッセージが表示されます。“Ｙ”キーを押下するとメモリ消去を開始し、消去終了すると
“Ｅｒａｓｅ　ｄｏｎｅ”メッセージ表示で消去が正常に終了します。
“Ｎ”キーを押下するとメモリは消去されず次のコマンド待ち状態になります。

【P】: ﾌﾟﾛｸﾞﾗﾑﾀﾞｳﾝﾛｰﾄﾞ（XMODEM)

■ 説明

プログラムを書き換えを行います。

■ フォーマット

P<CR>

■ 例

３．３項のハイパータミナルによるプログラム書き換え例を参照してください。

【ERR】: 最後に発生したｴﾗｰの表示

■ 説明

エラー発生時の最後のエラー状態を表示します。

■ フォーマット

ERR<CR>

■ 例

```
>ERR<CR>
8011 Timeout error
>
```

プログラム転送時にタイムアウトが発生した場合

【D】: ﾒﾓﾘｰﾀﾞﾝﾌﾟ

■ 説明

指定されたメモリアドレスの内容を表示します。

■ フォーマット

Ｄ<CR>

■ 例

```
>D<CR>
Display Address (0-0x0FFFFFF) >010000

010000   7A 07 00 0A 00 00 01 00 6D F6 0F F6 5E 01 35 06   z.......m...^.5.
010010
   ・
   ・
   ・
Display Address (0-0x0FFFFFF) >
```

“Ｄ”コマンドを入力すると
“Ｄｉｓｐｌａｙ　Ａｄｄｒｅｓｓ（０－０ｘＦＦＦＦＦＦ）＞”メッセージが表示さ、表示するメモリ内容のアドレスの入力待ちになります。アドレスの入力は０～ＦＦＦＦＦＦのＨＥＸ（１６進）コードで指定します。ここでは“０１００００”と入力します。するとメモリアドレスの０１００００番地～０１００ＦＦ番地（ＨＥＸ）の２５６バイトの内容が表示されます。
２５６バイト表示後“Ｄｉｓｐｌａｙ　Ａｄｄｒｅｓｓ（０－０ｘＦＦＦＦＦＦ）＞” メッセージが表示され入力待ちになります。
“Ｅｎｔｅｒ”キーを押下すると次のアドレスのメモリ内容２５６バイトが表示されます。
終了するときは“ＥＳＣ”キーを押下してください。

【M】:　ﾒﾓﾘｰ内容変更

■ 説明
指定されたメモリアドレスの内容を１バイト単位で書き換えをおこないます。
■ フォーマット
M<CR>
■ 例

```
>M<CR>
Display Address (0-0x0FFFFFF) >e0000
0E0000 : [AD] >>10

0E0001 : [EC] >>
```

“M”コマンドを入力すると
“Ｄｉｓｐｌａｙ　Ａｄｄｒｅｓｓ（０－０ｘＦＦＦＦＦＦ）＞”メッセージが表示さ、書き換えるメモリのアドレスの入力待ちになります。アドレスの入力は０～ＦＦＦＦＦＦのＨＥＸ（１６進）コードで指定します。ここでは“ｅ００００”と入力します。“０Ｅ００００：［ＡＤ］＞＞”と０Ｅ００００番地の内容“AD”がＨＥＸコードで表示され、次に変更するデータの入力待ちになります。この例ではＨＥＸコードの“１０”と入力し、“Ｅｎｔｅｒ”キーを押下します。

【FD】: ﾃﾞｰﾀﾒﾓﾘｰの読出

■ 説明

NAND 型フラッシュメモリーに格納されたデータを５２８バイト単位で読み出します。
NAND 型フラッシュメモリーはページ単位で読み書きする構造になっています。
8M バイト（64MBSupport）の場合　最大ページ数は 16884 ページとなります。
AT8012 では、先頭ページを 0 ページとしております。

■ フォーマット

FD<CR>

■ 例

```
>FD<CR>
Display Page (0-16383) >1<CR>
010000   7A 07 00 0A 00 00 01 00 6D F6 0F F6 5E 01 35 06   z.......m...^.5.
010010
    ・
    ・
    ・
Display Page (0-16383) >
```

表示が終了すると再びページ表示ページを要求するプロンプトが出ます。
引き続き次のページを読む場合は<CR>のみ、他のページを読み出すときは数値を入力してください。
<ESC>キーで終了します。

【FE】: ﾃﾞｰﾀﾒﾓﾘｰの消去

■ 説明

NAND 型フラッシュメモリーに格納されたデータの消去を行います。
NAND 型フラッシュメモリーの消去単位はブロック単位になっています。
8M バイト（64MBSupport）の場合　1 ブロックは 16 ページで 1024 ブロックとなっています。
AT8012 では、先頭ブロックを 0 ページとしております。

■ フォーマット

FE<CR>

■ 例

```
>FE<CR>
Erase Page [0-1024] (1024 all page) >0<CR>
Flash memory Erase complite !

Erase Page [0-1024] (1024 all page) >
```

消去が終了すると消去が成功した旨を伝えるメッセージを表示し、再びページ消去ブロックを要求するプロンプトが出ます。
引き続きブロックを消去する場合、ブロックを指定する数値を入力してください。
<ESC>キーで終了します。
数値で 1024 を指定するとすべてのブロックを消去します

【FW】: ﾃﾞｰﾀﾒﾓﾘｰへの書込ﾃｽﾄ

■ 説明

NAND 型フラッシュメモリーの１ページを指定したデータで FILL します。

■ フォーマット

FW<CR>

■ 例

```
>FW<CR>
Write page (0-16383) >0
Write data (00-FF) >AA
Flash memory write complite !
Write page (0-16383) >
```

コマンドを入力すると、書き込むページを要求するプロンプトが表示されます。
書込を行うページを指定してください。
続いて FILL するデータを要求するプロンプトが表示されますので、データを１６進数で入力してください。
書込が終了すると書込が成功した旨を伝えるメッセージを表示し、再びページ書き込むページを要求するプロンプトが出ます。
引き続きブロックを書込する場合、書込をするページの数値を入力してください。
<ESC>キーで終了します。

【FMEBM】: ﾃﾞｰﾀﾒﾓﾘｰのｴﾗｰﾁｪｯｸ ＆ ｴﾗｰﾃｰﾌﾞﾙの作成

■ 説明

NAND 型フラッシュメモリー不良ブロックを検出し、検出結果をテーブル化し、ROM の記録します。

NAND 型フラッシュメモリーはスペック上数ブロックの不良を持っています。その不良ブロックを予め検出し、そのブロックを使用しないことを目的としています。

このテーブルはユーザープログラム ROM 領域にかかれますので、ユーザープログラムが確定しプログラムを転送した後で実行してください。ユーザープログラムの消去を実行すると一緒に消去されます。(ユーザープログラム消去時はチップイレーズを実行する為)

■ フォーマット

FMEBM<CR>

■ 例

```
>FMEBM<CR>
Flash memory Error Block Check Start ? (Y/N)
Flash memory ERASE start
Step 1.  A5 Write
Step 2.  A5 Read & check
Write error A5... At page 5503
Write error A5... At page 6591
Write error A5... At page 7551
Write error A5... At page 13871
Write error A5... At page 14703
Step 3.  Erase
Step 4.  5A Write
Step 5.  5A Read & check
Step 6.  Erase
Flash memory check END
```

>

この例では数箇所エラーが発見されています。
このコマンドを実行すると最長３０分位かかりますのでご注意願います。

【FMEBR】: ﾃﾞｰﾀﾒﾓﾘｰのｴﾗｰ情報読

■ 説明

FMEBM コマンドで作成したテーブルの内容を読み出します。

■ フォーマット

FMEBR<CR>

■ 例

```
>FMEBR<CR>
343 block error !!
 411 block error !!
 471 block error !!
 866 block error !!
 918 block error !!
 >
```

【FMEBC】: ﾃﾞｰﾀﾒﾓﾘｰのｴﾗｰ情報ｸﾘｱ

■ 説明

FMEBM コマンドで作成したテーブルの内容を消去します。

■ フォーマット

FMEBC<CR>

【SETRTC】: ｶﾚﾝﾀﾞｰ時計の設定

■ 説明

内蔵 RTC の設定を行います。

■ フォーマット

SETRTC <CR>

■ 例

日付を 2002 年 5 月 8 日　水曜日　午後 9 時 19 分 30 秒に設定します

```
>SETRTC<CR>
Date (YYMMDD) >020508<CR>
Week select
  0:SUN. 1:MON. 2:TUE. 3:WED. 4:THU. 5:FRI 6:SAT
select (0-6) >3<CR>
Time (HHMMSS) >211930<CR>
>
```

年は西暦で入力してください。
時刻は２４時間制で設定してください。

【GETRTC】: ｶﾚﾝﾀﾞｰ時計の表示

■ 説明
内蔵 RTC の読み出しを行います。

■ フォーマット
GETRTC <CR>

■ 例
>GETRTC<CR>
2002/05/08 WED 21:19:30
>

【XMODEM】: ﾃﾞｰﾀﾒﾓﾘｰの読出 (XMODEM)

このコマンドは使用しないでください。

【SHUT】: ｽﾀﾝﾊﾞｲﾓｰﾄﾞ検

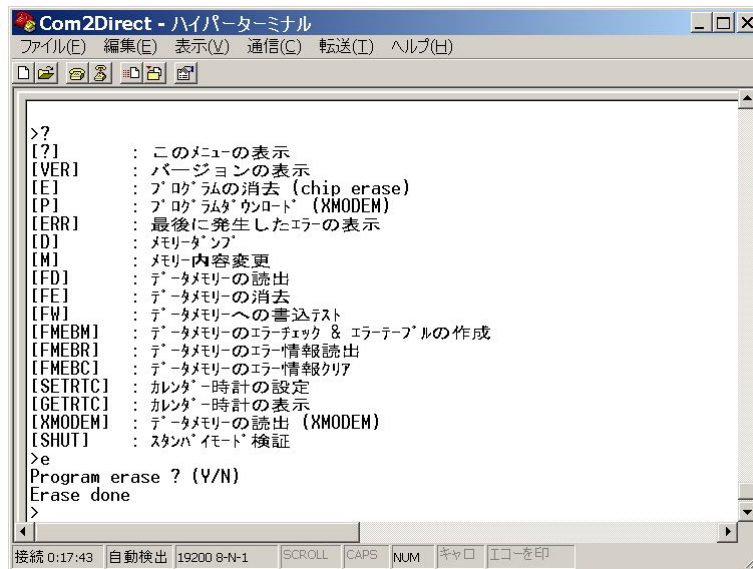
■ 説明
低消費電力状態に遷移します。
このコマンドは復帰しません。電源を入れなおしてください。

■ フォーマット
SHUT <CR>

■ 例
>SHUT<CR>

３．３ ハイパータミナルによるプログラム書き換え例

プログラムを書き換える場合まず、メモリーを消去する必要がありますので、メニューの“E”コマンドを入力し、“Y”キーを入力するとメモリーが消去されます。(画面 F)

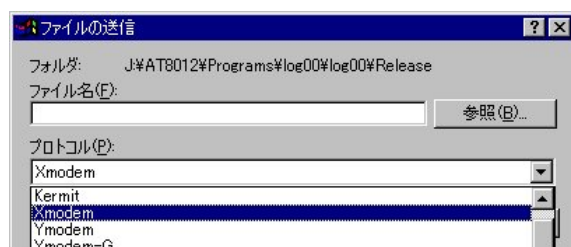
画面Ｆ

次に、“P”コマンドを入力し、“Y”を入力するとプログラムの転送待ち状態になりますので、ハイパーターミナルのプログラムの転送を行います。(画面ＧからＫ)

画面Ｇ

```
Com2Direct - ハイパーターミナル
ファイル(F) 編集(E) 表示(V) 通信(C) 転送(T) ヘルプ(H)
[?]       : このﾒﾆｭｰの表示
[VER]     : バージョンの表示
[E]       : ﾌﾟﾛｸﾞﾗﾑの消去 (chip erase)
[P]       : ﾌﾟﾛｸﾞﾗﾑﾀﾞｳﾝﾛｰﾄﾞ (XMODEM)
[ERR]     : 最後に発生したｴﾗｰの表示
[D]       : ﾒﾓﾘｰﾀﾞﾝﾌﾟ
[M]       : ﾒﾓﾘｰ内容変更
[FD]      : ﾃﾞｰﾀﾒﾓﾘｰの読出
[FE]      : ﾃﾞｰﾀﾒﾓﾘｰの消去
[FW]      : ﾃﾞｰﾀﾒﾓﾘｰへの書込ﾃｽﾄ
[FMEBM]   : ﾃﾞｰﾀﾒﾓﾘｰのｴﾗｰﾁｪｯｸ & ｴﾗｰﾃｰﾌﾞﾙの作成
[FMEBR]   : ﾃﾞｰﾀﾒﾓﾘｰのｴﾗｰ情報読出
[FMEBC]   : ﾃﾞｰﾀﾒﾓﾘｰのｴﾗｰ情報ｸﾘｱ
[SETRTC]  : ｶﾚﾝﾀﾞｰ時計の設定
[GETRTC]  : ｶﾚﾝﾀﾞｰ時計の表示
[XMODEM]  : ﾃﾞｰﾀﾒﾓﾘｰの読出 (XMODEM)
[SHUT]    : ｽﾀﾝﾊﾞｲﾓｰﾄﾞ検証
>e
Program erase ? (Y/N)
Erase done
>p
Write Program ? (Y/N)
Push ESC key to CANCEL
§§§§
接続 0:24:46 自動検出 19200 8-N-1 SCROLL CAPS NUM キャロ エコーを印
```

画面Ｈ

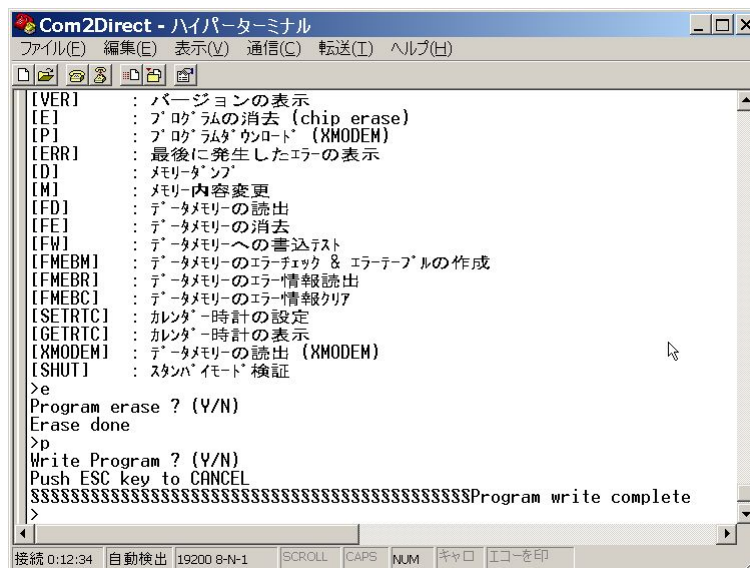
ハイパーターミナルのファイル転送を選択

画面Ｉ

プロトコルをＸｍｏｄｅｍに設定し、転送するファイル名称を入力する。
転送するファイルはモトローラＨＥＸフォーマットファイルを指定してください。

画面Ｊ

Xmodem ファイル送信 - ComDirect
送信中: J:¥AT8012¥Programs¥log00¥log00¥Release¥log00.mot
パケット: 32 エラー チェック: Checksum
再試行: 0 再試行の回数: 0
最新のエラー:
ファイル: 4K / 93K
経過時間: 00:00:03 残り: 00:01:13 スループット: 1237 cps
キャンセル cps/bps(C)

データ転送中の画面で、パケットの数値がしばらくたっても上がらない場合はキャンセルを押してから、ＥＳＣキーを入力してください。
その後、“Ｌ”コマンドを入力し、エラーが発生していないかチェックしてください。

画面Ｋ

正常に終了すると上記画面になります。

第５章　ＲＴＣの使い方

　ＡＴ８０１２は株式会社リコー社製　アラーム機能付きＲＴＣ　「ＲＳ５Ｃ３１７Ｂ」を搭載しております。
　　ＲＴＣの機能詳細については同社発行のマニュアルを参照してください。

ＣＰＵとの接続はマイコンのポート及び割り込み端子（ＮＭＩ）に接続されています。
（ブロック図参照）

制御方法については、付録のＣソースを参照してください。

第６章　ＮＡＮＤフラッシュメモリの使い方

ＮＡＮＤ型フラッシュメモリーは株式会社東芝社製「ＴＣ5864ＦＴ」または同等品を搭載しております。このメモリーの使用方法は通常メモリーと異なり、アドレスバスがありません。読み出し、書き込みアドレスはコマンドで与えます。詳細は製品のマニュアルを参照してください。

ＮＡＮＤ型フラッシュメモリにアクセスする場合、データバスの配線がねじれて接続されているため、コマンドを発行する際に注意が必要です。また、ステータスチェックについても同様です。
フラッシュメモリーアクセスのサンプルソースを付録に添付いたします。

フラッシュメモリーのデータバスの接続

データバス	NAND のデータバス
D7	D0
D6	D7
D5	D1
D4	D6
D3	D2
D2	D5
D1	D3
D0	D4

第７章　低消費電力モード

ＡＴ８０１２で低消費電力状態にするには、Ｈ８マイコンのソフトウェアスタンバイモードに遷移させせます。　Ｈ８マイコンのコントロールレジスタのスタンバイビットをセットし、スリープ状態にします。

このとき、使用していないＨ８の端子は出来るだけ出力状態にし、Ｌｏを出力するようにしてください。

データバス及びアドレスバスについては、フローティング状態になってしまいますが、約 100μＡまで消費電力が低下します。

これ以上の低下を望むのであれば、コネクタを介しバスなどをプルアップ（プルダウン）してください。

この処置をすると約 10μＡまで低下します。

ＡＴ８０１２用の１６ビットＡ／Ｄ変換モジュール（別売）を使用すると、上記状態にすることができます。

第８章　プログラミングの方法

通常のＲＯＭ化する方法を用いコンパイル等を実行してください。プログラムの転送はモトローラＳフォーマットしか受け付けません。モトローラＳフォーマットのレコードタイプＳ１、Ｓ２、Ｓ３のアドレスモードに対応しています。

７．１ ベクタテーブル設定方法

本来のベクタテーブルはＨ８内蔵のＲＯＭ内にあるため、書き換えが出来ないので、ＲＯＭ内のベクタテーブルはＲＡＭ（Ｈ８内蔵ＲＡＭ）の固定アドレス（ＦＦ７１０Ｈ番地～ＦＦ８１０Ｈ番地）に割り当てられています。

ＲＡＭのＦＦ７１０Ｈ番地～ＦＦ８１０Ｈ番地にＪＭＰコード＋割込処理のアドレスを設定することにより、ベクタテーブルとして機能させるようにコーディングしてください。

以下にサンプルソースコードを示します。

```
#pragma section INTJMP
unsigned long                _intjmp[64] ;                // Interrupt Jump table
#pragma section


void
SetIntrAddr(
            int              intno,
            Iunsigned long   func
)
{
UW                  jmpcode ;

/* Interrupt jump table initialize       */

            jmpcode = func | 0x5a000000L;

            _intjmp[intno] = jmpcode ;

}
```

ソースの説明

#progma section INTJMP

この行はこれ移行の行セクションを意図的に指定する命令です。

リンケージエディターでこのセクションをＦＦ７１０ H 番地に指定してください。

SetIntrAddr 関数

この関数は割り込みベクター番号と、その関数のアドレスを指定することにより、ＲＡM内の擬似ベクターテーブルに登録を行うものです。

ソース内の 0x5a000000L は H8 マイコンのジャンプコードです。

第９章　仕　様

8.１　ＡＴ８０１２仕様

表１　　AT8012 仕様

CPU	日立製作所製　H8/3042
動作周波数	6.144MHz
データ記憶容量	4M バイト（NAND 形フラッシュメモリー）
RTC	カレンダー機能、バッテリーバックアップ付き
外部割り込み	1 チャンネル
シリアルポート	1 チャンネル、調歩同期
I/O ポート	10 チャンネル
アナログ入力	分解能 10 ビット、入力 8 チャンネル
最高計測周期	1msec
電源	＋6V～＋12V
バックアップ電源	＋+1.5～3V
消費電流（平均値）	25mA（通常動作時）
	30mA（A/D 変換時）
	75mA（フラッシュメモリー書き込み時）
	15mA（スリープ時）
	300uA（ソフトウェアスタンバイ時）
最大許容消費電流	100mA（周辺回路を含む許容される最大消費電流）
本体寸法	別紙参照

8.２　電気的特性

表２　　AT8012 絶対最大定格

項　　目	記 号	定　格　値	単 位
電源電圧	Vdd	12.0	V
レギュレータ出力電圧	Vout	Vss－0.3　～　Vdd＋0.3	V
レギュレータ許容損失	Pd	500	mW
アナログ電源電圧	Avdd	－0.3　～　＋7.0	V
リファレンス電源電圧	Avref	－0.3　～　AVdd＋0.3	V
アナログ入力電圧	AVan	－0.3　～　AVdd＋0.3	V
動作温度	Topr	－20　～　＋70	℃
保存温度	Tstg	－40　～　＋90	℃

湿度は、0～90％　ただし、結露しないこと

8.３　ＡＴ８０１２Ａ／Ｄ変換特性

表３　　AT8012A/D変換特性

項　　目	min	typ	max	単位
分解能	10	10	10	bit
変換時間	－	－	21.8	uS
アナログ入力容量	－	－	20	pF
許容信号源インピーダンス	－	－	10	kΩ
非直線性誤差	－	－	±3.0	LSB
オフセット誤差	－	－	±2.0	LSB
フルスケール誤差	－	－	±2.0	LSB
量子化誤差	－	－	±0.5	LSB
絶対精度	－	－	±4.0	LSB

条件：Vout＝5.0V±10%、AVdd＝5.0V±10%、AVref＝4.5V～AVdd、Vss＝AVss＝0V

付　録

以下に掲載されたソースコードは、ご要望に応じてＥ－ｍａｉｌなどで送付いたします。
注意）コンパイル時は最適化をしないでください。

全般のＤＥＦＩＮＥ文

```
typedef char   B;                       /* 符号付き8ビット整数 */
typedef short  H;
typedef        int          INT ;
typedef long   W;                       /* 符号付き 32 ビット整数 */
typedef unsigned char       UB;                 /* 符号無し8ビット整数 */
typedef unsigned short      UH;                 /* 符号無し 16 ビット整数 */
typedef unsigned            UINT;
typedef unsigned long       UW;                 /* 符号無し 32 ビット整数 */
typedef char   VB;                      /* タイプ不定データ(8ビットサイズ)*/
typedef long   VW;                      /* タイプ不定データ(32 ビットサイズ)*/
typedef void   *VP;                     /* タイプ不定データへのポインタ */
typedef void   (*FP)(void);             /* プログラムのスタートアドレス一般 */
typedef short  VH;

typedef enum { FALSE = 0, TRUE } boolean;

#define        FOREVER                  for(;;)

#define        ON                               1
#define        OFF                              0
```

1.　フラッシュメモリーアクセスのためのＣソースコード

フラッシュメモリーアクセスの為のＤＥＦＩＮＥ文

```
#define        FM_MAX_PAGE         16384
#define        FM_MAX_BLK          1024
#define        FM_MAX_BYTE         528
#define        FM_BLK_PAGE         16

typedef        struct  {
               UB                               :4 ;
               UB          CSE                  :1 ;
               UB          CLE                  :1 ;
               UB          ALE                  :1 ;
               UB          BSY                  :1 ;
} TFmemCtl ;

typedef        union   {
   struct      {
     UB        PAS         :1 ;     /* 0: Pass          1: Fail          b0     */
     UB        PRO         :1 ;     /* 0: Protect   1: Not protect  b7     */
     UB        B1          :1 ;
     UB        BSY         :1 ;     /* 0: Busy          1: Not busy      b6     */
     UB        B2          :1
     UB        SUS         :1 ;     /* 0: Not Suspended 1: Suspend       b5     */
     UB        B3          :1 ;
     UB        B4          :1 ;
   } bi ;
   UB          b ;
} UFmemStatus ;

#define        FmCntl              (*(TFmemCtl *)(0xFFFFC7))
#define        FmData              (*(UB *)(0xA0000))



#define        FM_AWRITE           0x40          // Command 0x80
#define        FM_READ1            0x00          // Command 0x00
#define        FM_READ2            0x11          // Command 0x50
#define        FM_RESET            0xFF          // Command 0xFF
#define        FM_PROG             0x01          // Comamnd 0x10
#define        FM_BERASE           0x14          // Command 0x60
#define        FM_STATE            0x15          // Command 0x70
#define        FM_RESUME           0x51          // Command 0xD0
```

フラッシュメモリーの読み出し

```
void
FmRead(
		UH			page,
		UB			*data
)
{
UShort		*x ;
int			i ;
UB			sts ;
UH			pg ;

		FOREVER	{
			if( FmCntl.BSY )
				break ;
		}
		pg = FmDtCnv( page ) ;
		x = (UShort *)&pg ;
		FmCntl.ALE = FmCntl.CLE = FmCntl.CSE = 0 ;
		FmCntl.CLE = 1 ;
		FmData = FM_READ1 ;
		FmCntl.CLE = 0 ;
		FmCntl.ALE = 1 ;
		FmData = 0 ;
		FmData = x->b.l ;
		FmData = x->b.h ;
		FmCntl.ALE = 0 ;
		sts = 0 ;
		FOREVER	{
			if( FmCntl.BSY )
				break ;
		}
		for( i = 0; i < FM_MAX_BYTE; i++ )		{
			data[i] = FmData ;
		}
		FmCntl.CSE = 1 ;
}
```

コマンドコード変換（マイコンのデータバスがフラッシュメモリーのバスとねじれて接続されているためのコンバージョン）

```
UH
FmDtCnv(
		UH		page
)
{
UH			ret ;
UShort		*s ;
UShort		*d ;
char		txt[30] ;

		s = (UShort *)&page ;
		d = (UShort *)&ret ;


		d->bbi.h.b7 = s->bbi.h.b0 ;
		d->bbi.h.b6 = s->bbi.h.b7 ;
		d->bbi.h.b5 = s->bbi.h.b1 ;
		d->bbi.h.b4 = s->bbi.h.b6 ;
		d->bbi.h.b3 = s->bbi.h.b2 ;
		d->bbi.h.b2 = s->bbi.h.b5 ;
		d->bbi.h.b1 = s->bbi.h.b3 ;
		d->bbi.h.b0 = s->bbi.h.b4 ;

		d->bbi.l.b7 = s->bbi.l.b0 ;
		d->bbi.l.b6 = s->bbi.l.b7 ;
		d->bbi.l.b5 = s->bbi.l.b1 ;
		d->bbi.l.b4 = s->bbi.l.b6 ;
		d->bbi.l.b3 = s->bbi.l.b2 ;
		d->bbi.l.b2 = s->bbi.l.b5 ;
		d->bbi.l.b1 = s->bbi.l.b3 ;
		d->bbi.l.b0 = s->bbi.l.b4 ;

		return( ret ) ;
}
```

フラッシュメモリーの書込み

```
boolean
FmWrite(
		UH				page,
		UB				*data
)
{
UShort					*x ;
int							i ;
UFmemStatus			sts ;
boolean					ret ;
UH							pg ;

		FOREVER	{
				if( FmCntl.BSY )
						break ;
		}
		pg = FmDtCnv( page ) ;
		x = (UShort *)&pg ;
		FmCntl.ALE = FmCntl.CLE = FmCntl.CSE = 0 ;
		FmCntl.CLE = 1 ;
		FmData = FM_AWRITE ;
		FmCntl.CLE = 0 ;
		FmCntl.ALE = 1 ;
		FmData = 0 ;
		FmData = x->b.l ;
		FmData = x->b.h ;
		FmCntl.ALE = 0 ;
		FOREVER
		{
				if( FmCntl.BSY )
						break ;
		}
		for( i = 0; i < FM_MAX_BYTE; i++ )
				FmData = data[i] ;
		FmCntl.CLE = 1 ;
		FmData = FM_PROG ;
		FmCntl.CLE = 0 ;
		FOREVER
		{
				if( FmCntl.BSY )
						break ;
		}
		FOREVER
		{
				FmCntl.CLE = 1 ;
				FmData = FM_STATE ;
				FmCntl.CLE = 0 ;
				sts.b = FmData ;
				if( !sts.bi.BSY )	continue ;
				if( !sts.bi.PAS )	ret = TRUE ;
				else			ret = FALSE ;
				break ;
		}
		FmCntl.CSE = 1 ;
		return ret ;
}
```

フラッシュメモリーのリセット

```
void
FmReset(
		void
)
{
		FmCntl.ALE = 0 ;
		FmCntl.CLE = 0 ;
		FmCntl.CSE	= 0 ;

		FOREVER	{
				if( FmCntl.BSY )
						break ;
		}

		FmCntl.CLE = 1 ;
		FmData = FM_RESET ;
		FmCntl.CLE = 0 ;

		FOREVER	{
				if( FmCntl.BSY )
						break ;
		}

		FmCntl.CSE = 1 ;
}
```

フラッシュメモリー消去

```
boolean
FmErase(
        UH                      block
)
{
UH                                          xx, blk ;
UShort                  *x ;
UFmemStatus         sts ;
boolean                 ret ;

        x = (UShort *)&xx ;
        blk = block << 4 ;
        xx = FmDtCnv( blk ) ;
        FmCntl.ALE = 0 ;
        FmCntl.CLE = 0 ;
        FmCntl.CSE   = 0 ;
        FOREVER     {
                if( FmCntl.BSY )
                        break ;
        }
        FmCntl.CLE = 1 ;
        FmData = FM_BERASE ;
        FmCntl.CLE = 0 ;
        FmCntl.ALE = 1 ;
        FmData = x->b.l ;
        FmData = x->b.h ;
        FmCntl.ALE = 0 ;
        FmCntl.CLE = 1 ;
        FmData = FM_RESUME ;
        FmCntl.CLE = 0 ;
        FOREVER     {
                if( FmCntl.BSY )
                        break ;
        }
        FOREVER
        {
                FmCntl.CLE = 1 ;
                FmData = FM_STATE ;
                FmCntl.CLE = 0 ;
                sts.b = FmData ;
                if( !sts.bi.BSY )
                        continue ;
                if( !sts.bi.PAS )
                        ret = TRUE ;
                else
                        ret = FALSE ;
                break ;
        }
        FmCntl.CSE = 1 ;
        return ret ;
}
```

2.　ＲＴＣアクセスのためのＣソースコード

ＲＴＣアクセスの為のＤＥＦＩＮＥ文

```
#define    RTC_CYC_OFF       0x00
#define    RTC_CYC_L         0x01
#define    RTC_CYC_1K        0x02
#define    RTC_CYC_05S       0x03
#define    RTC_CYC_1S        0x08
#define    RTC_CYC_10S       0x09
#define    RTC_CYC_1M        0x0A
#define    RTC_CYC_10M       0x0B
#define    RTC_CYC_1H        0x0C
#define    RTC_CYC_1D        0x0D
#define    RTC_CYC_1W        0x0E
#define    RTC_CYC_1MO       0x0F

#define    RTC_SEC01_REG     0x00     // 1 秒ｶｳﾝﾀ
#define    RTC_SEC10_REG     0x01     // 10 秒ｶｳﾝﾀ
#define    RTC_MIN01_REG     0x02     // 1 分ｶｳﾝﾀ
#define    RTC_MIN10_REG     0x03     // 10 分ｶｳﾝﾀ
#define    RTC_HOR01_REG     0x04     // 1 時ｶｳﾝﾀ
#define    RTC_HOR10_REG     0x05     // 10 時ｶｳﾝﾀ
#define    RTC_DAY01_REG     0x08     // 1 日ｶｳﾝﾀ
#define    RTC_DAY10_REG     0x09     // 10 日ｶｳﾝﾀ
#define    RTC_MON01_REG     0x0A     // 1 月ｶｳﾝﾀ
#define    RTC_MON10_REG     0x0B     // 10 月ｶｳﾝﾀ
#define    RTC_YEA01_REG     0x0C     // 1 年ｶｳﾝﾀ
#define    RTC_YEA10_REG     0x0D     // 10 年ｶｳﾝﾀ

#define    RTC_WEK_REG       0x06     // 曜日ｶｳﾝﾀ
#define    RTC_CYC_REG       0x07     // 割込周期設定ﾚｼﾞｽﾀ
#define    RTC_CTL01_REG     0x0E     // 制御ﾚｼﾞｽﾀ 1
#define    RTC_CTL02_REG     0x0F     // 制御ﾚｼﾞｽﾀ 2

#define    RTC_AWEK01_REG    0x00     // ｱﾗｰﾑ曜日ﾚｼﾞｽﾀ 1
#define    RTC_AWEK02_REG    0x01     // ｱﾗｰﾑ曜日ﾚｼﾞｽﾀ 2
#define    RTC_AMIN01_REG    0x02     // ｱﾗｰﾑ 1 分ﾚｼﾞｽﾀ
#define    RTC_AMIN10_REG    0x03     // ｱﾗｰﾑ 10 分ﾚｼﾞｽﾀ
#define    RTC_AHOR01_REG    0x04     // ｱﾗｰﾑ 1 時ﾚｼﾞｽﾀ
#define    RTC_AHOR10_REG    0x05     // ｱﾗｰﾑ 10 時ﾚｼﾞｽﾀ

#define    RTC_TMR_REG       0x09     // ﾀｲﾏｰﾚｼﾞｽﾀ
#define    RTC_32K_REG       0x0A     // 32KHz 制御ﾚｼﾞｽﾀ
```

ＲＴＣの完全初期化

ＲＴＣのバッテリーバックアップ状態を監視し、初期電源投入であれば各レジスタを初期化します。

```
boolean
RtcInit(
		void
)
{
char			dat ;

		P4.DR.BIT.B5 = 1 ;				// Chip Enable set

		RtcDataGet( RTC_CTL01_REG, &dat ) ;

		if( !(dat & 0x02) )
			return FALSE ;

		dat = 0 ;
		RtcDataSet( RTC_CYC_REG, 0 ) ;				// 割込み周期ﾚｼﾞｽﾀ ｸﾘｱ
		RtcDataSet( RTC_CTL02_REG, 3 ) ;			// ﾊﾞﾝｸ 1 設定
		RtcDataSet( RTC_AHOR10_REG, 0 ) ;			// ALE bit clear
		RtcDataSet( RTC_CTL01_REG, 3 ) ;			// ADJ bit set

		FOREVER
		{
			RtcDataGet( RTC_CTL01_REG, &dat ) ;
			if( !( dat & 0x01 ) )
				break ;
		}

		RtcDataSet( RTC_CTL01_REG, 0 ) ;			//
		RtcDataSet( RTC_CTL02_REG, 9 ) ;			// 24 時間制設定

		RtcDataSet( RTC_SEC01_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_SEC10_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_MIN01_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_MIN10_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_HOR01_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_HOR10_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_DAY01_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_DAY10_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_MON01_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_MON10_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_YEA01_REG, 0 ) ;			// 設定
		RtcDataSet( RTC_YEA10_REG, 0 ) ;			// 設定

		RtcDataSet( RTC_WEK_REG, 0 ) ;				// 設定

		P4.DR.BIT.B5 = 0 ;				// Chip Enable set
		P4.DR.BIT.B6 = 1 ;

		return TRUE ;
}
```

日付時刻の読み出し

ＲＴＣのレジスタよりデータを読み出し、ＢＣＤで値を返します

```
void
RtcGetDateTime(
		char		*dt					// yy,mm,dd,hh,mm,ss,ww
)
{
char			x ;
int					i;

		FOREVER
		{
				P4.DR.BIT.B5 = 1 ;
				RtcDataSet( RTC_CTL02_REG, 0x09 ) ;

				RtcDataGet( RTC_CTL01_REG, &x ) ;
				if( x & 0x01 )
				{
						for( i = 0; i < 500; i++ )
						{
								P4.DR.BIT.B5 = 0 ;
						}
						continue ;
				}
				else
						break ;
		}

		RtcDataGet( RTC_SEC01_REG, &x ) ;		// Get
		dt[5] = x & 0x0F ;
		RtcDataGet( RTC_SEC10_REG, &x ) ;		// Get
		dt[5] |= ((x << 4) & 0xF0) ;
		RtcDataGet( RTC_MIN01_REG, &x ) ;		// Get
		dt[4] = x & 0x0F ;
		RtcDataGet( RTC_MIN10_REG, &x ) ;		// Get
		dt[4] |= ((x << 4) & 0xF0) ;
		RtcDataGet( RTC_HOR01_REG, &x ) ;		// Get
		dt[3] = x & 0x0F ;
		RtcDataGet( RTC_HOR10_REG, &x ) ;		// Get
		dt[3] |= ((x << 4) & 0xF0) ;
		RtcDataGet( RTC_DAY01_REG, &x ) ;		// Get
		dt[2] = x & 0x0F ;
		RtcDataGet( RTC_DAY10_REG, &x ) ;		// Get
		dt[2] |= ((x << 4) & 0xF0) ;
		RtcDataGet( RTC_MON01_REG, &x ) ;		// Get
		dt[1] = x & 0x0F ;
		RtcDataGet( RTC_MON10_REG, &x ) ;		// Get
		dt[1] |= ((x << 4) & 0xF0) ;
		RtcDataGet( RTC_YEA01_REG, &x ) ;		// Get
		dt[0] = x & 0x0F ;
		RtcDataGet( RTC_YEA10_REG, &x ) ;		// Get
		dt[0] |= ((x << 4) & 0xF0) ;
		RtcDataGet( RTC_WEK_REG, &x ) ;				// Get
		dt[6] = x ;

		P4.DR.BIT.B5 = 0 ;
		P4.DR.BIT.B6 = 1 ;
}
```

ＲＴＣへの日付・時刻の書込
設定はＢＣＤ値で与えます

```
void
RtcSetDateTime(
		char		*dt						// yy,mm,dd,hh,mm,ss,ww
)
{
char			x ;
int				i;

		FOREVER
		{
			P4.DR.BIT.B5 = 1 ;
			RtcDataSet( RTC_CTL02_REG, 0x09 ) ;

			RtcDataGet( RTC_CTL01_REG, &x ) ;
			if( x & 0x01 )
			{
				for( i = 0; i < 500; i++ )
				{
					P4.DR.BIT.B5 = 0 ;
				}
				continue ;
			}
			else
				break ;
		}


		x = dt[5] & 0x0F ;
		RtcDataSet( RTC_SEC01_REG, x ) ;		// 設定
		x = (dt[5] >> 4) & 0x0F ;
		RtcDataSet( RTC_SEC10_REG, x ) ;		// 設定

		x = dt[4] & 0x0F ;
		RtcDataSet( RTC_MIN01_REG, x ) ;		// 設定
		x = (dt[4] >> 4) & 0x0F ;
		RtcDataSet( RTC_MIN10_REG, x ) ;		// 設定

		x = dt[3] & 0x0F ;
		RtcDataSet( RTC_HOR01_REG, x ) ;		// 設定
		x = (dt[3] >> 4) & 0x0F ;
		RtcDataSet( RTC_HOR10_REG, x ) ;		// 設定

		x = dt[6] & 0x0F ;
		RtcDataSet( RTC_WEK_REG, x ) ;			// 設定

		x = dt[2] & 0x0F ;
		RtcDataSet( RTC_DAY01_REG, x ) ;		// 設定
		x = (dt[2] >> 4) & 0x0F ;
		RtcDataSet( RTC_DAY10_REG, x ) ;		// 設定

		x = dt[1] & 0x0F ;
		RtcDataSet( RTC_MON01_REG, x ) ;		// 設定
		x = (dt[1] >> 4) & 0x0F ;
		RtcDataSet( RTC_MON10_REG, x ) ;		// 設定

		x = dt[0] & 0x0F ;
		RtcDataSet( RTC_YEA01_REG, x ) ;		// 設定
		x = (dt[0] >> 4) & 0x0F ;
		RtcDataSet( RTC_YEA10_REG, x ) ;		// 設定

		P4.DR.BIT.B5 = 0 ;
		P4.DR.BIT.B6 = 1 ;
}
```

アラームの設定

```
void
RtcSetAlm(
		char		*dt						// hh,mm,ww
)
{
char				x ;


		P4.DR.BIT.B5 = 1 ;
		RtcDataSet( RTC_CTL02_REG, 0x0B ) ;

		x = dt[2] & 0x0F ;
		RtcDataSet( RTC_AWEK01_REG, x ) ;			// 設定
		x = (dt[2] >> 4) & 0x0F ;
		RtcDataSet( RTC_AWEK02_REG, x ) ;			// 設定

		x = dt[1] & 0x0F ;
		RtcDataSet( RTC_AMIN01_REG, x ) ;			// 設定
		x = (dt[1] >> 4) & 0x0F ;
		RtcDataSet( RTC_AMIN10_REG, x ) ;			// 設定

		x = dt[0] & 0x0F ;
		RtcDataSet( RTC_AHOR01_REG, x ) ;			// 設定
		x = (dt[0] >> 4) & 0x0F ;
		RtcDataSet( RTC_AHOR10_REG, x ) ;			// 設定

		RtcDataSet( RTC_CTL02_REG, 0x09 ) ;

		P4.DR.BIT.B5 = 0 ;
		P4.DR.BIT.B6 = 1 ;

}
```

アラーム設定内容の読み出し

```
void
RtcGetAlm(
		char		*dt								// hh,mm,ww
)
{
char				x ;

		P4.DR.BIT.B5 = 1 ;
		RtcDataSet( RTC_CTL02_REG, 0x0B ) ;

		RtcDataGet( RTC_AWEK01_REG, &x ) ;			// Get
		dt[2] = x & 0x0F ;
		RtcDataGet( RTC_AWEK02_REG, &x ) ;			// Get
		dt[2] |= ((x << 4) & 0xF0) ;

		RtcDataGet( RTC_AMIN01_REG, &x ) ;			// Get
		dt[1] = x & 0x0F ;
		RtcDataGet( RTC_AMIN10_REG, &x ) ;			// Get
		dt[1] |= ((x << 4) & 0xF0) ;

		RtcDataGet( RTC_AHOR01_REG, &x ) ;			// Get
		dt[0] = x & 0x0F ;
		RtcDataGet( RTC_AHOR10_REG, &x ) ;			// Get
		dt[0] |= ((x << 4) & 0xF0) ;

		RtcDataSet( RTC_CTL02_REG, 0x09 ) ;

		P4.DR.BIT.B5 = 0 ;
		P4.DR.BIT.B6 = 1 ;
}
```

ＲＴＣレジスタ読み出し（シリアル->パラレル変換）

```
void
RtcDataGet(
		char		addr,			//読み出しアドレス
		char		*dt			//データ格納領域
)
{
UB			x ;
int			i ;


		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 1 ;				// R/W Read Set
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 1 ;				// AD Set
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 0 ;				// DT Clear
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		x = 0x08 ;
		for( i = 0; i < 4; i++ )
		{
			P4.DR.BIT.B6 = 1 ;				// SCLK Hi
			if( addr & x )
				P4.DR.BIT.B7 = 1 ;			// SIO
			else
				P4.DR.BIT.B7 = 0 ;
			x >>= 1 ;
			P4.DR.BIT.B6 = 1 ;				// SCLK Hi
			P4.DR.BIT.B6 = 0 ;				// SCLK Lo
			P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		}

		P4.DDR = 0x7E ;
		for( i = 0; i < 4; i++ )
		{
			P4.DR.BIT.B6 = 1 ;
			P4.DR.BIT.B6 = 1 ;
			P4.DR.BIT.B6 = 0 ;
			P4.DR.BIT.B6 = 0 ;
		}


		*dt = 0 ;
		for( i = 0; i < 4; i++ )
		{
			P4.DR.BIT.B6 = 1 ;
			P4.DR.BIT.B6 = 1 ;
			*dt <<= 1 ;
			*dt += P4.DR.BIT.B7 ;
			P4.DR.BIT.B6 = 0 ;
			P4.DR.BIT.B6 = 0 ;
		}

		P4.DDR = 0xFE ;
}
```

ＲＴＣレジスタ書込（パラレル->シリアル変換）

```
void
RtcDataSet(
		char		addr,				//書込みアドレス
		char		dt				// 書込みデータ
)
{
UB				x ;
int				i ;

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 0 ;				// R/W Write Set
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 1 ;				// AD Set
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 0 ;				// DT Clear
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		x = 0x08 ;
		for( i = 0; i < 4; i++ )
		{
			P4.DR.BIT.B6 = 1 ;				// SCLK Hi
			if( addr & x )
				P4.DR.BIT.B7 = 1 ;			// SIO
			else
				P4.DR.BIT.B7 = 0 ;
			x >>= 1 ;
			P4.DR.BIT.B6 = 1 ;				// SCLK Hi
			P4.DR.BIT.B6 = 0 ;				// SCLK Lo
			P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		}


		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 0 ;				// R/W Write Set
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 0 ;				// AD Clear
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B7 = 1 ;				// DT Set
		P4.DR.BIT.B6 = 1 ;				// SCLK Hi
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		P4.DR.BIT.B6 = 0 ;				// SCLK Lo

		x = 0x08 ;
		for( i = 0; i < 4; i++ )
		{
			P4.DR.BIT.B6 = 1 ;				// SCLK Hi
			if( dt & x )
				P4.DR.BIT.B7 = 1 ;			// SIO
			else
				P4.DR.BIT.B7 = 0 ;
			x >>= 1 ;
			P4.DR.BIT.B6 = 1 ;				// SCLK Hi
			P4.DR.BIT.B6 = 0 ;				// SCLK Lo
			P4.DR.BIT.B6 = 0 ;				// SCLK Lo
		}

}
```